# CHƯƠNG 1 ÔN LẠI KIẾN THỨC VỀ JAVASCRIPT

## Arrow function trong ES6

Là một phương thức rút gọn dành cho hàm, sử dụng mũi tên “=>”

Ví dụ:

* Đây là một hàm bình thường để in ra một đoạn console

function logger (log) {

console.log(log);

}

logger("Hello");

* Đây là sử dụng arrow function để rút gọn

const logger = (log) => {

console.log(log);

}

logger("Hello");

* Hoặc có thể rút gọn dấu ngoặc đơn nếu chỉ có 1 tham số

const logger = log => {

console.log(log);

}

logger("Hello");

* Đây là dạng tối giản hơn: Rút gọn dấu ngoặc nhọn và return, mặc định sau dấu “=>” là return, và thường sử dụng cho 1 lệnh đơn giản

const logger = (log) => console.log(log);

logger("Hello");

* Cũng như cách trên nhưng là một khối lệnh, khối lệnh này in ra object a và b, sẽ có 2 phép gán nên phải sử dụng ngoặc đơn

const logger = (a, b) => ({a: a, b: b});

console.log(logger(2, 3));

Tuy nhiên thì arrow function không có context riêng. Context được hiểu như là ngữ cảnh hay phạm vi của đối tượng. Xem ví dụ sau để hiểu

* Tạo một object, có thể thấy được khi gọi this.name thì this ở đây được hiểu như là đối tượng gọi đến getName (tức là course), nếu sử dụng function ở đây thì sẽ in ra được từ “Javacript”

const course = {

name: "Javascript",

getName: function () {

return this.name;

}

}

console.log(course.getName());

* Nhưng nếu sử dụng arrow function thì sẽ in ra undefined vì arrow function không có context riêng nên không thể gọi được name, vậy nên trong trường hợp này không sử dụng arrow function

const course = {

name: "Javascript",

getName: () => {

return this.name;

}

}

console.log(course.getName());

## Module trong ES6

Là việc tác các thành phần quan trọng ra các nơi riêng, tập hợp các file nhỏ lẻ

Import là nạp vào, Export là xuất ra. Để sử dụng import và export buộc file js phải có kiểu dữ liệu là module. VD: <script type="module" src="./script.js"></script>

Mỗi file js chỉ được phép export default 1 lần duy nhất, có thể có nhiều export thông thường

## Enhanced object literals trong ES6

Ví dụ: Tạo một đối tượng như sau

var name = 'JavaScript'

var price = 1000;

var course = {

name: name,

price: price

};

Có thể viết đơn giản như sau, Sử dụng để định nghĩa key: value cho object

var name = 'JavaScript'

var price = 1000;

var course = {

name,

price,

};

Sử dụng để định nghĩa method cho object

var name = 'JavaScript'

var price = 1000;

var course = {

name,

price,

getName() {

return name;

}

};

Sử dụng để định nghĩa key cho object dưới dạng biến, biến đặt trong ngoặc vuông. Khi in ra tên mới sẽ là ReactJS chứ không còn là JavaScript nữa

var name = 'JavaScript'

var price = 1000;

var course = {

[name]: 'ReactJS',

price,

};

## Rest paramater trong ES6

Rest là lấy ra những phần còn lại

VD:In ra 1,2,3,4,5 vì nó sẽ lấy params và params đang là toàn bộ số truyền vào

function logger(...params){

console.log(params);

}

logger(1,2,3,4,5)

Tuy nhiên nếu sử dụng một biến khác như sau, thì params sẽ loại bỏ phần tử đầu tiên cho biến đó, phần còn lại sẽ là params: 2,3,4,5

function logger(a, ...params){

console.log(params);

}

logger(1,2,3,4,5)

## Spread trong ES6

Ví dụ: nối 2 mảng, spread sẽ bỏ đi dấu [] đối với mảng và {} đối với obj. Nếu trùng thuộc tính sẽ lấy cái sau

var arr1 = [1, 2]

var arr2 = [3, 4, 5]

var arr3 = [...arr1, ...arr2]

console.log(arr3);

## Detructuring trong ES6

Phân rã cấu trúc

VD: In ra 2 biến a và c, là 1 và 3. Số 2 sẽ không được định nghĩa, bỏ qua bằng cách phẩy luôn và không ghi gì

var arr = [1, 2, 3]

var [a, , c] = arr

console.log(a, c);

VD2: Khi gán biến phải trùng với thuộc tính của object, có thể đổi bằng dấu “:”. Nếu đặt thuộc tính không có trong object, có thể cho nó một giá trị mặc định nếu thuộc tính đó không tồn tại, nếu tồn tại sẽ lấy giá trị đã tồn tại đó

var course = {

name: 'Javascript',

price: 1000,

children: {

name: 'ReactJS'

}

};

var {name: parenName, price, children: {name}, description = ‘default’} = course

console.log(parenName, price, name) // Javascript 1000 ReactJS

# TỔNG QUAN VỀ REACT

## Khái niệm React

React là một thư viện JavaScript phổ biến để xây dựng các giao diện người dùng dựa trên thành phần, có thể tái sử dụng cho các trang web hoặc ứng dụng.

React kết hợp HTML với chức năng JavaScript thành ngôn ngữ đánh dấu của riêng nó được gọi là JSX. React cũng giúp dễ dàng quản lý luồng dữ liệu xuyên suốt ứng dụng.

React là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook để xây dựng giao diện người dùng cho các ứng dụng web đơn trang và đa trang. Nó cho phép phát triển các thành phần UI động, có thể tái sử dụng và dễ bảo trì trong các ứng dụng web lớn và phức tạp. React sử dụng mô hình hướng thành phần, nghĩa là các thành phần UI được xây dựng bằng cách kết hợp các thành phần nhỏ hơn. React cũng được sử dụng rộng rãi để phát triển các ứng dụng di động bằng React Native.

Hiện tại chỉ có 2 phiên bản chính của React là:

* ReactJS: là phiên bản chính của React, được sử dụng để phát triển các ứng dụng web đơn trang và đa trang.
* React Native: là một phiên bản của React được sử dụng để phát triển các ứng dụng di động cho iOS và Android.
* React VR: (không phổ biến) dành cho phát triển các ứng dụng thực tế ảo và React 360 cho phát triển các ứng dụng thực tế ảo trên web.

## SPA/MPA

SPA là Single-Page Application: Ứng dụng một trang: Facebook, Google, Twitter, … SPA tiếp cận hiện đại hơn, không tải lại trang trong quá trình sử dụng

MPA là Multi-Page Application: Ứng dụng nhiều trang, là cách tiếp cận cổ điển, tải lại trang trong quá trình sử dụng

# PHÂN TÍCH CÁC THÀNH PHẦN REACT

## Tạo phần tử bằng JavaScript (DOM)

**document.createElement()**

**Cách 1: Sử dụng để tạo một element**

Tạo một thẻ h1 gán cho hằng số h1, hiện tại h1 đang là một object

const h1 = document.createElement('h1');

Có thể thêm đoạn văn bản bên trong h1 bằng innerText

h1.innerText = 'Hello';

Tuy nhiên h1 vẫn chưa được render ra màn hình, mới chỉ trong phần code console thôi, để render, chúng ta có thể thử hàm appendChild() để cho h1 vào con của thẻ khác

document.body.appendChild(h1);

Như vậy thẻ h1 đã là con của thẻ body và sẽ được hiển thị trên màn hình

Để thêm thuộc tính cho thẻ h1 ta làm như sau, thêm id, className, style

h1.id = 'heading';

h1.className = 'title title2';

h1.style.color = 'red';

**Cách 2: Sử dụng Object.assgin để gán cho nhiều dữ liệu**

Lưu ý tên các thuộc tính sẽ khác một chút so với CSS, dấu ‘-‘ trong CSS được loại bỏ và chữ tiếp theo được viết hoa, ví dụ:

Object.assign(h1.style, {

color: 'red',

backgroundColor: '#ccc'

})

## Gọi phần tử bằng JavaScript (DOM)

**document.querySelector()**

Lấy ra phần tử đầu tiên của HTML với bộ chọn CSS chỉ định, tuy nhiên chỉ lấy phần tử đầu tiên nó tìm thấy

document.querySelector('#heading') // lấy ra id có tên là heading

document.querySelector('.title') // lấy ra class đầu tiên có tên là title

## Thêm React vào website

Các bước để thêm React vào website

Link: <https://react.dev/learn/add-react-to-an-existing-project>

Dán vào thẻ head của html:

<script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>

## Tạo phần tử bằng React

React.createElement()

Tạo một thẻ h1

const h1 = React.createElement(‘h1’);

Thêm lớp, thêm id.

Cấu trúc trong ngoặc (type, props, children)

type: tên thẻ, ...

prop: được đặt trong ngoặc nhọn { } bao gồm các thuộc tính, nếu 1 thuộc tính thì không cần ngoặc nhọn

children: inner bên trong thẻ có thể là text, object, array, ....

const h1 = React.createElement('h1', {

title: 'Hello',

className: 'heading'

}, 'Hello world')

## React DOM

Dùng để render ra màn hình chính, cú pháp thêm phần reactDOM như sau:

<script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>

<script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>

Cấu trúc ReactDOM:

const container = getElementById(‘id of contaner’);

const root = ReactDOM.createRoot(container);

root.render(element);

Trong đó:

* Element: biến chứa thẻ con
* Container: biến chứa thẻ cha

VD1: Tạo thẻ 1 list gồm ReactJS và JavaScript, tại file index.js

import React from "react";

import ReactDOM from "react-dom/client";

import reportWebVitals from "./reportWebVitals";

const root = ReactDOM.createRoot(document.getElementById("root"));

const ul = React.createElement(

"ul",

null,

React.createElement("li", null, "ReactJS"),

React.createElement("li", null, "JavaScript")

);

root.render(

<React.StrictMode>

{ul}

</React.StrictMode>

);

reportWebVitals();

## JSX

JSX là JavaScript XML, XML khác với HTML nhưng về cơ bản 2 cái này khá giống nhau, nhưng để chạy được JSX cần có babel để chuyển đổi cú pháp ES6 trở thành ES5 để phù hợp cho nhiều loại trình duyệt

Để sử dụng babel ta khai báo như sau dưới ReactDOM

<script src="https://unpkg.com/@babel/standalone@7.15.0/babel.min.js"></script>

Và phải định dạng file script với type=”text/babel”:

<script type="text/babel" src="./script.js"></script>

Trong JSX sẽ chỉ được tạo và render 1 thẻ trong 1 lần tạo, muốn tạo nhiều phải tạo và render nhiều, để khắc phục được điều đó thì chúng ta phải wrap chúng lại trong 1 thẻ cha, tuy nhiên nếu không muốn tạo ra một thẻ cha (ví dụ như div) thì có thể sử dụng thẻ <React.Fragment></React.Fragment> sẽ không sinh ra thẻ cha nhưng vẫn wrap được

Có thể sử dụng viết tắt <></> thay cho React.Fragment trong React

## React element types

Khi chúng ta tạo một phần tử: React.createElement(types, props, children);

Type có thể là string, function hay class. Chẳng hạn chuỗi ‘h1’ sẽ tạo ra thẻ h1

Đối với function trong React sẽ phải viết hoa biến function chữ cái đầu. Function được gọi là React Component vì có thẻ tách rời ra sử dụng nhiều nơi, còn bên trong sẽ gọi là React Element.

## Props

Các cú pháp của html vẫn sẽ được sử dụng trong React Element nhưng sẽ có một vài thay đổi, ví dụ : for trong html sẽ là forcus vào id đối tượng nhưng trong React lại là vòng lặp nên for được đổi thành htmlFor, class trong React là lớp nên khi muốn sử dụng class html phải đổi thành className.

Props là đối số của component, props có thể là bất cứ dữ liệu gì.

const data = [

{

id: 1,

title: "This is title 1",

content: "This is content 1",

},

{

id: 2,

title: "This is title 2",

content: "This is content 2",

},

];

const Item = function (props) {

return (

<div>

<h1 style={{ color: "red" }}>{props.title}</h1>

<h2 style={{ color: "orange" }}>{props.content}</h2>

</div>

);

};

function App() {

return (

<div className="App">

{data.map((item, index) => (

<Item key={index} title={item.title} content={item.content}></Item>

))}

</div>

);

}

export default App;

## DOM events

DOM events là các sự kiện xảy ra khi chúng ta tương tác với đối tượng, ví dụ click, change, copy, ...

const Item = function (props) {

const handleClick = () => {

alert(props.title);

};

return (

<div>

<h1 style={{ color: "red" }}>{props.title}</h1>

<h2 style={{ color: "orange" }}>{props.content}</h2>

<button onClick={handleClick}>Click me</button>

</div>

);

};

## Component linh hooạt

Trong JSX sẽ không in ra kiểu dữ liệu boolean, undefined, null. Có thể sử dụng điều này để kiểm tra

VD:

const Form = {

Input() {

return <input type="text" />;

},

Checkbox() {

return <input type="checkbox" />;

},

};

function App() {

const Component = Form["Checkbox"];

return (

<div className="App">

<Component></Component>

</div>

);

}

export default App;

VD: Có thể sử dụng biến đặt biệt chilren để hiểu nội dung bên trong là children

const Button = ({ children }) => {

return <button>{children}</button>;

};

function App() {

return (

<div className="App">

<Button>Click me</Button>

</div>

);

}

export default App;

## NodeJS

NodeJS là JavaScript runtime là một môi trường độc lập để chạy JavaScript, thông thường ta sử dụng trình duyệt để chạy JavaScript. NodeJS không liên quan tới trình duyệt hoàn toàn chạy độc lập.

Sử dụng NodeJS dùng cho cả font-end cả back-end. Trong bài React này NodeJS sẽ được sử dụng để làm máy chủ

## Webpack

Trong React App sẽ có tích hợp webpack rồi nên sẽ không cần cài webpack, phần này sẽ nói về lý thuyết webpack.

Webpack giúp module hóa font-end

## React Webpack

* Bước 1: tạo thư mục chứa project
* Bước 2: bật terminal và trỏ và thư mục chứa project (vd: react-webpack), gõ npm init
* Bước 3: gõ enter liên tiếp để cài mọi thứ mặc định, sau đó chúng ta sẽ có file package.json
* Bước 3: tại terminal gõ:

npm install webpack webpack-cli --save-dev

* Bước 4: gõ tiếp:

npm i react@latest react-dom@latest

* Bước 5: gõ tiếp:

npm install @babel/core babel-loader @babel/preset-env @babel/preset-react --save-dev

* Bước 6: tạo file public/index.html và nhấn ! tab để tạo ra 1 template html, thêm thẻ div id root ở body
* Bước 7: tạo file src/index.js

import React from "react";

import ReactDOM from "react-dom/client";

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(

<React.StrictMode>

<div>

<h1>Hello world</h1>

</div>

</React.StrictMode>

);

* Bước 8: cấu hình webpack, tại terminal gõ:

npm install css-loader style-loader --save-dev

* Bước 9: đặt file webpack.config.js ở thư mục gốc, ngang hàng với package.json

const path = require("path");

module.exports = {

entry: "./src/index.js", // Dẫn tới file index.js ta đã tạo

output: {

path: path.join(\_\_dirname, "/build"), // Thư mục chứa file được build ra

filename: "bundle.js" // Tên file được build ra

},

module: {

rules: [

{

test: /\.js$/, // Sẽ sử dụng babel-loader cho những file .js

exclude: /node\_modules/, // Loại trừ thư mục node\_modules

use: ["babel-loader"]

},

{

test: /\.css$/, // Sử dụng style-loader, css-loader cho file .css

use: ["style-loader", "css-loader"]

}

]

},

// Chứa các plugins sẽ cài đặt trong tương lai

plugins: [

]

};

* Bước 10: Tạo file .babelrc ở thư mục gốc

{

"presets": [

"@babel/preset-env",

"@babel/preset-react"

]

}

* Bước 11: Tại file package.json thêm đoạn sau vào cuối “script” (trong script nhưng ở vị trí cuối)

"start": "webpack --mode development --watch",

"build": "webpack --mode production"

* Bước 12: Tại terminal gõ

npm start

lúc này 1 file build/bundle.js đã được tự động build. Lúc này terminal đang có dạng không thể gõ được

* Bước 13: Tại ternimal gõ Ctrl + C để thoát, gõ tiếp

npm install html-webpack-plugin --save-dev

* Bước 14: Tại file webpack.config.js thêm lệnh này trước module.exports

const HtmlWebpackPlugin = require("html-webpack-plugin");

Và thêm đoạn sau trong mảng plugin

new HtmlWebpackPlugin({

template: "./public/index.html"

})

* Bước 15: tại terminal gõ

npm start

lúc này sẽ tạo ra 1 file build/index.html và dòng script trong đó, tuy nhiên nó sẽ được đặt trên head, mặc dù vậy nhưng nó vẫn tải nhưng đợi html chạy xong nó mới chạy vì có từ khóa defer, làm vậy không cần cài thủ công script trong cuối thẻ body mà vẫn chạy sau cùng

* Bước 16: Ctrl + C ternminal để thoát, gõ

npm install webpack-dev-server --save-dev

* Bước 17: Sửa file webpack.config.js

Xóa dòng: "start": "webpack --mode development --watch",

Thay bằng: "start": "webpack-dev-server --mode development --open --hot",

* Bước 18: Tại terminal gõ

npm start

Cập nhật phiên bản react và react-dom mới nhất: npm i react@latest react-dom@latest

Chuyển về phiên bản cụ thể (vd 17.0.2): npm i react@17.0.2 [react-dom@17.0.2](mailto:react-dom@17.0.2)

## Create React App

npm create vite@latest tên\_dự\_án -- --template react-ts

Cấu hình vite.config.ts

import { defineConfig } from "vite";

import react from "@vitejs/plugin-react";

import path from "path";

// https://vite.dev/config/

export default defineConfig({

plugins: [react()],

resolve: {

alias: {

"~": path.resolve(\_\_dirname, "src"),

},

},

});

## Hook

Hooks là những method, hàm được viết sẵn và cung cấp bởi ReactJS, mỗi hàm có 1 tính năng, 1 trường hợp cụ thể để sử dụng. Tác dụng để gắn vào các component.

Hooks chỉ dùng cho function component, không dùng được cho class component. Và ngoài function component không thể sử dụng được ở bất cứ đâu

### useState

Khi muốn dữ liệu thay đổi thì giao diện tự động cập nhật (render lại theo dữ liệu).

* Component được re-render sau khi được setState
* Initial state chỉ dùng cho lần đầu
* Set state với callback
* Initial state với callback
* Set state là thay thế state bằng giá trị mới

VD: Khi click vào nút increase thì sẽ tăng 1 giá trị

import { useState } from "react";

function App() {

const [counter, setCounter] = useState(0);

const handleIncrement = () => {

setCounter(counter + 1);

};

return (

<div className="app">

<button onClick={handleIncrement}>current Value: {counter}</button>

</div>

);

}

export default App;

Lưu ý state chỉ thực hiện cập nhật giá trị khi kết thúc function gọi đến nó:

// counter = 1

const handleIncrease = () => {

setCounter(counter + 1); // 1 + 1 = 2

setCounter(counter + 1); // 1 + 1 = 2

}

// counter = 2

Bản chất 2 lần setCounter đều lấy giá trị cũ của counter là 1 và 2 lần gán bằng 2 chứ không phải là 3 vì hàm chưa kết thúc nên counter chưa được set 2

Tuy nhiên nếu muốn thao tác với bộ nhớ đệm của nó, tức là giá trị nó sắp được nhận nhưng muốn lấy nó để thao tác xử lý luôn

// counter = 1

const handleIncrease = () => {

setCounter(prevState => prevState + 1); // 1 + 1 = 2

setCounter(prevState => prevState + 1); // 2 + 1 = 3

}

// counter = 3

### useEffect

Sử dụng trong trường hợp muốn thực hiện các side effect như: Update DOM, Call API, DOM event, Clean up, …

useEffect sẽ thực hiện sau khi render

Cú pháp: useEffect(callback, dependency);

* useEffect(callback): Luôn thực hiện sau mỗi lần render
* useEffect(callback, []): Chỉ gọi 1 lần khi component mounted
* useEffect(callback, dependency): Gọi 1 lần đầu sau khi mounted, và từ lần 2 trở đi sẽ theo dõi dependency thay đổi sẽ được chạy

Trong useEffect lưu ý việc clean up mỗi khi dùng các event chẳng hạn như các hàm: setInterval, setTimeOut, async, EventListener, subcribeEvent, … Các hàm này sẽ chạy ngay cả khi unMounted.

Clean up sẽ là mục return của useEffect

VD:

useEffect(() => {

return () => {

if (intervalId.current) {

clearInterval(intervalId.current);

}

};

}, [intervalId]);

Cách thức hoạt động của Cleanup trong useEffect như sau:

Lần 1: Không cleanup

Lần 2: Cleanup của lần 1

Lần 3: Cleanup của lần 2

Cứ như vậy khi chạy lần nào thì sẽ dọn dẹp của lần trước, điều này giúp tránh rò rỉ bộ nhớ.

### useLayoutEffect

Về cơ bản thì giống với useEffect, tuy nhiên thực hiện callback trước khi render. Nhưng useLayoutEffect không khuyến khích được sử dụng vì:

* Logic phức tạp sẽ chậm quá trình render gây UX kém
* Đối với SSR (Server side render) không hỗ trợ
* Khó tái sử dụng

Chỉ nên sử dụng trong trường hợp đo đạc kích thước phần tử, tự động focus, scrolling

### useRef

Là một object có duy nhất 1 thuộc tính là current. Giá trị lưu trong current không thay đổi dù có render nhiều lần. Khi nó thay đổi thì không làm re-render

Có thể gắn trực tiếp lên element JSX: Sử dụng trong việc thao tác với event element (focus, onClick, onChange, đo kích thước, tích hợp thư viện không phải react.

Giải quyết được vấn đề closure, tương tác nhiều với setInterval và setTimeout

### memo

memo là memoization là HOC (High-Order Component) không phải là hook. Đóng gói một component chỉ re-render khi props thay đổi. Sử dụng toán tử === để so sánh.

Cú pháp:

import { memo } from "react";

function Memo1() {

return <div></div>;

}

export default memo(Memo1);

VD:

import { memo } from "react";

function Memo1({ text }: { text: string }) {

return <div>{text}</div>;

}

export default memo(Memo1);

Hữu ích khi:

* Component con nhận props là object/array hay function stable.
* Tránh re-render thừa khi parent re-render mà props không đổi.

### useCallback

Là một hook luôn đi cùng với memo cho phép bạn ghi nhớ một hàm callback để:

* Tránh tạo ra hàm mới sau mỗi lần re-render (trừ khi dependency thay đổi), từ đó ngăn các component con phụ thuộc vào callback đó bị re-render thừa.
* Miễn là các giá trị trong mảng phụ thuộc không thay đổi thì nó sẽ không tạo mới.

Xem xét ví dụ sau:

Content.tsx

import { memo } from "react";

interface Props {

onIncrease: () => void;

}

function Content({ onIncrease }: Props) {

console.log("Re-render");

return (

<div>

<p>Hello World!</p>

<button onClick={onIncrease}>Click me</button>

</div>

);

}

export default memo(Content);

Index.tsx

import { useState } from "react";

import Content from "@/components/Content";

function Index() {

const [count, setCount] = useState(0);

const onIncrease = () => {

setCount(count + 1);

};

return (

<div>

<Content onIncrease={onIncrease} />

<h1>{count}</h1>

</div>

);

}

export default Index;

Theo ví dụ trên mặc dù mỗi lần click có thay đổi count nhưng là thay đổi cả Content, bị render lại mặc dù hàm không thay đổi.

Khá nghịch lý với memo vì memo sẽ không render lại khi deps không đổi.

Xét ví dụ sau:

const a = () => {}

const b = () => {}

Mặc dù hàm a và b là giống nhau nhưng lại khởi tạo 2 object khác nhau tham chiếu đến vùng nhớ khác nhau nên khi so sánh === thì lại thấy vùng tham chiếu hàm a khác vùng tham chiếu hàm b nên props đã có sự thay đổi và Content bị render lại.

React cung cấp useCallback giúp chặn việc tạo ra hàm mới nếu props không thay đổi (so sánh bằng Object.is)

import { useState, useCallback } from "react";

import Content from "@/components/Content";

function Index() {

const [count, setCount] = useState(0);

const onIncrease = useCallback(() => {

setCount((prevCount) => prevCount + 1);

}, []);

return (

<div>

<Content onIncrease={onIncrease} />

<h1>{count}</h1>

</div>

);

}

export default Index;

### useMemo

Ghi nhớ kết quả của 1 logic, nếu deps không thay đổi thì sẽ không thực hiện việc tính toán lại.

Tóm tắt:

* memo: cache component render
* useCallback: cache function
* useMemo: cache kết quả tính toán

### useReducer

### useContext

### useImperativeHandle

## CSS

### CSS inline

Sử dụng ngay trong thẻ DOM element, không khuyến khích vì đây là code cứng, khó debug sau này

<h1 style={{color: 'red'}}>Hello</h1>

### CSS external

import ‘./style.css’

Tuy nhiên nó sẽ hiển thị trên trình duyệt là dạng internal <style>...</style> và nhiều file css sẽ có nhiều thẻ style

Và thông thường khi run build (npm run build) cho production sẽ gộp lại làm 1 file css điều này gây xung đột giữa các file css, css sau sẽ gán đè lên file trc nếu trùng className

### CSS module

Là phương pháp cấu trúc hóa CSS trong ứng dụng web tránh xung đột giữa phạm vi các file CSS

Để sử dụng CSS module, cần sửa file css có dạng .module.css, Ví dụ App.module.css và nó sẽ export 1 object tên là styles, lưu ý đặt tên className theo CamelCase vì sẽ tiện cho sử dụng js, trong js không được dùng dấu –

VD: header-bottom => headerBottom

import styles from ‘./App.module.css

object style sẽ chứa các class làm key và value sẽ là 1 đoạn chuỗi cấu tạo từ nhiều phần

VD:

.module.css:

.paragraph{

color: red;

}

.heading{

font-size: 16px;

background-color: #ccc;

}

Lúc này object style sẽ có dạng như sau:

styles {

heading: “Paragraph\_heading\_\_1kGV-“,

paragraph: Paragraph\_paragraph\_\_ThmR99”

}

Paragraph: Thư mục cha chứa file .module.css

heading: tên className

\_\_: ký tự ngăn cách

1kGV-: mã hóa đường dẫn hatch B64

**Để sử dụng className trong file js ta làm như sau:**

import styles from ‘./App.module.css’

<p className = {styles.heading}>Hello</p>

Tuy nhiên sẽ không sử dụng được các thẻ tag mặc định mà không gọi className, vì sẽ bị ảnh hưởng đến các file khác. Nếu dùng sass cấu trúc lồng nhau thì được

**Nếu dùng nhiều class**

VD1:

import styles from './style.module.css'

function App() {

return (

<div>

<h1 className={`${styles.heading} ${styles.content}`}>Hello</h1>

</div>

)

}

export default App;

VD2:

import styles from './style.module.css'

function App() {

return (

<div>

<h1 className={[styles.heading, styles.content].join(' ')}>Hello</h1>

</div>

)

}

export default App;

### Khuyên dùng

Sử dụng scss với module kết hợp (nếu chưa biết scss có thể dùng css):

npm install -D tailwindcss sass postcss autoprefixer @tailwindcss/postcss

npm install -D @tailwindcss/postcss

Tạo file: postcss.config.cjs

module.exports = {

plugins: {

"@tailwindcss/postcss": {},

autoprefixer: {},

},

};

Tạo file: tailwind.config.js

/\*\* @type {import('tailwindcss').Config} \*/

module.exports = {

content: ["./src/\*\*/\*.{js,ts,jsx,tsx}", "./index.html"],

theme: {

extend: {},

},

plugins: [],

};

Tạo 1 file css toàn cục: assets/css/index.css

Thêm 3 dòng sau trên đầu file:

@tailwind base;

@tailwind components;

@tailwind utilities;

Import vào file vào file main.tsx

import "./assets/css/index.css";

## Router

Giúp trang web có thể định tuyến qua lại giữa các trang trong cùng một dự án, cung cấp các chức năng hữu ích.

Cài đặt:

npm install react-router-dom

Tại file gốc của dự án: main.tsx, bao bọc toàn bộ dự án

import { BrowserRouter } from "react-router-dom";

…

<StrictMode>

<BrowserRouter>

<App />

</BrowserRouter>

</StrictMode>

Các component hữu ích trong "react-router-dom" bao gồm:

* Routes: là container quả lý các router, dùng để bao bọc các route (như thẻ ul)
* Route: xác định một định tuyến cho một đường dẫn cụ thể ánh xạ đến một component. Nhận các tham số props như (path, element, index: mặc định không không có đường dẫn nào khớp)
* Link: là component dùng để tạo các liên kết điều hướng người dùng đến các đường dẫn khác mà không reload toàn bộ trang. Thay thế thẻ a trong html mà vẫn giữ nguyên SPA. Sử dụng thuộc tính **to** nhận url để điều hướng

# CÀI ĐẶT NHANH

## Tạo dự án (sử dụng vite)

terminal

npm create vite@latest tên\_dự\_án -- --template react-ts

cd tên\_dự\_án

npm install --save-dev @types/node

## Cấu hình config giúp alias thư mục dễ dàng hơn

vite.config.ts

import { defineConfig } from "vite";

import react from "@vitejs/plugin-react";

import path from "path";

// https://vitejs.dev/config/

export default defineConfig({

plugins: [react()],

resolve: {

alias: {

"@": path.resolve(\_\_dirname, "./src"),

}

},

});

tsconfig.node.json và tsconfig.app.json đều thêm 2 dòng sau:

{

"compilerOptions": {

...

"baseUrl": ".",

"paths": {

"@/\*": ["src/\*"]

}

},

}

## Router

npm install react-router-dom

Tại file gốc của dự án: main.tsx, bao bọc toàn bộ dự án

import { BrowserRouter } from "react-router-dom";

…

<StrictMode>

<BrowserRouter>

<App />

</BrowserRouter>

</StrictMode>

## SCSS

terminal

npm install -D tailwindcss@latest @tailwindcss/vite

npm install postcss

npm install tailwindcss

vite.config.ts

import { defineConfig } from 'vite';

import react from '@vitejs/plugin-react';

import tailwindcss from '@tailwindcss/vite';

export default defineConfig({

plugins: [react(), tailwindcss()],

});

Tạo 1 file css toàn cục: assets/css/index.css

Thêm dòng sau trên đầu file:

@import "tailwindcss";

Import vào file vào file main.tsx

import "./assets/css/index.css";

## Normalize

Giúp css đồng nhất giữa các trình duyệt

npm install normalize.css

Tại main.tsx

import 'normalize.css';

## Prettier (tuỳ chọn)

npm install --save-dev prettier

Tạo: .prettierrc

{

"semi": true,

"singleQuote": false,

"tabWidth": 2,

"trailingComma": "es5"

}

Cài đặt trình IDE: tự động format bằng prettier